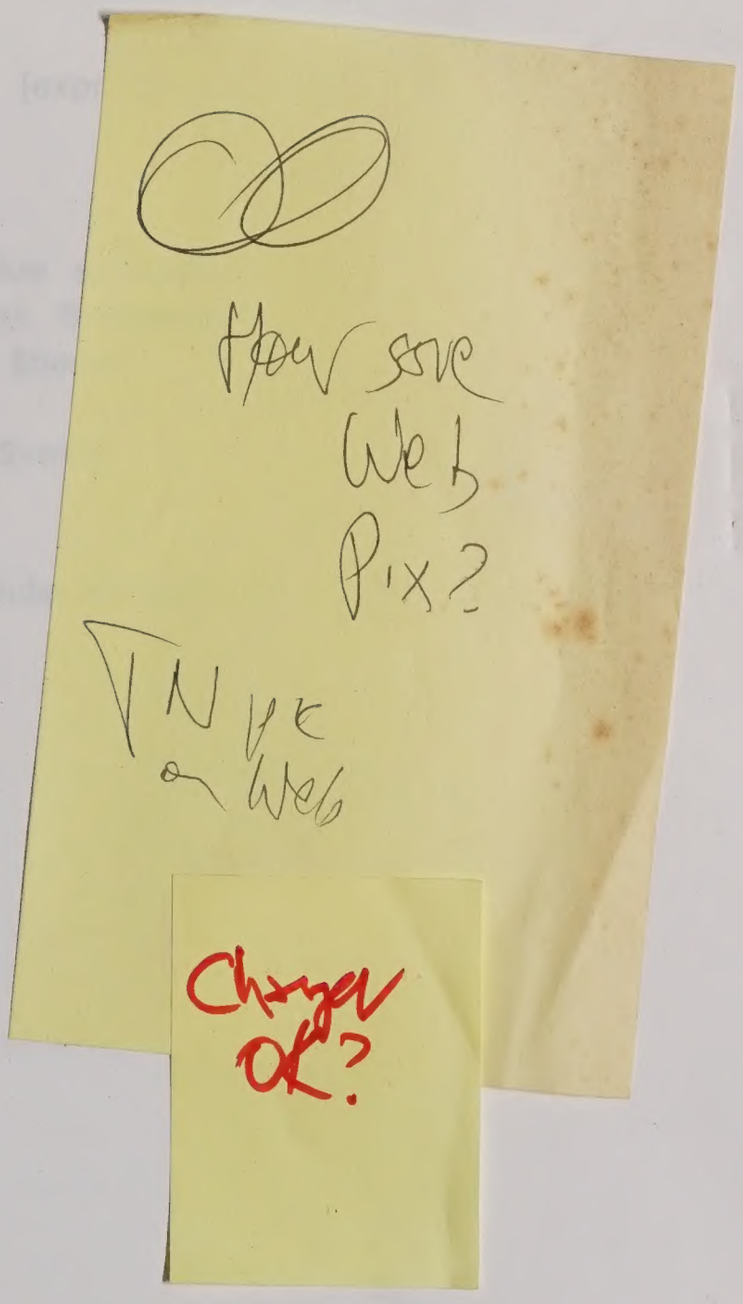
array.tcl
A program by Mr. Okidoki to:

```
proc print-array arrayName {
        upvar $arrayName a
        puts [array name a]
        foreach element [array name a] {
                puts [format "%s(%s) = %s" $arrayName $element
$a($element)]
        }
}

proc test-insert {n} {
        set list {}
        for {set i 0} {$i < $n} {incr i} {
                set list [linsert $list 0 $i]
        }
        return $list
}
```



com.tcl
A program by Mr. Okidoki to:

```
proc is-number {n} {
      regexp {^[0-9]+$|^-[0-9]+$} $n
}

proc get-attribute {attributes name separator} {
      set attributeList [split $attributes $separator]
      set index [lsearch $attributeList $name]
      if {$index < 0} {
            return ""
      } else {
            return [lindex $attributeList [expr $index + 1]]
      }
}

proc set-attribute {attributes name value separator} {
      set attributeList [split $attributes $separator]
      set index [lsearch $attributeList $name]
      if {$index < 0} {
            return  $attributes+$name+$value
      } else {
            set index [expr $index + 1]
            return [join [lreplace $attributeList $index $index $value] +]
      }
}

proc test-test {} {
      puts abcdefg
}
```

file.tcl
A program by Mr. Okidoki to:

```
set GenColumns 9
set GenZero 000000000
set GenFormat %9.9d

set SomeLargeNumber 999999999
# Is it large enough?

proc generation-zero {} {
	global GenZero
	return $GenZero
}
proc author-name {} {
	upvar #0 env(USER) authorName
	return $authorName
}

proc FILE:print {fileName} {
	set f [open $fileName r]
	while {! [eof $f]} {
		puts [gets $f]
	}
	close $f
}

proc generate-seq-no {} {
	global GenFormat
	set seqNoFileName .seqNo
	if [catch {set f [open $seqNoFileName r]} err] {
		set prev 0
	} else {
		set prev [gets $f]
		close $f
```

file.tcl
A program by Mr. Okidoki to:

```
set GenColumns 9
set GenZero 000000000
set GenFormat %9.9d

set SomeLargeNumber 999999999
# Is it large enough?

proc generation-zero {} {
	global GenZero
	return $GenZero
}

proc author-name {} {
	upvar #0 env(USER) authorName
	return $authorName
}

proc FILE:print {fileName} {
	set f [open $fileName r]
	while {! [eof $f]} {
		puts [gets $f]
	}
	close $f
}

proc generate-seq-no {} {
	global GenFormat
	set seqNoFileName .seqNo
	if [catch {set f [open $seqNoFileName r]} err] {
		set prev 0
	} else {
		set prev [gets $f]
		close $f
```

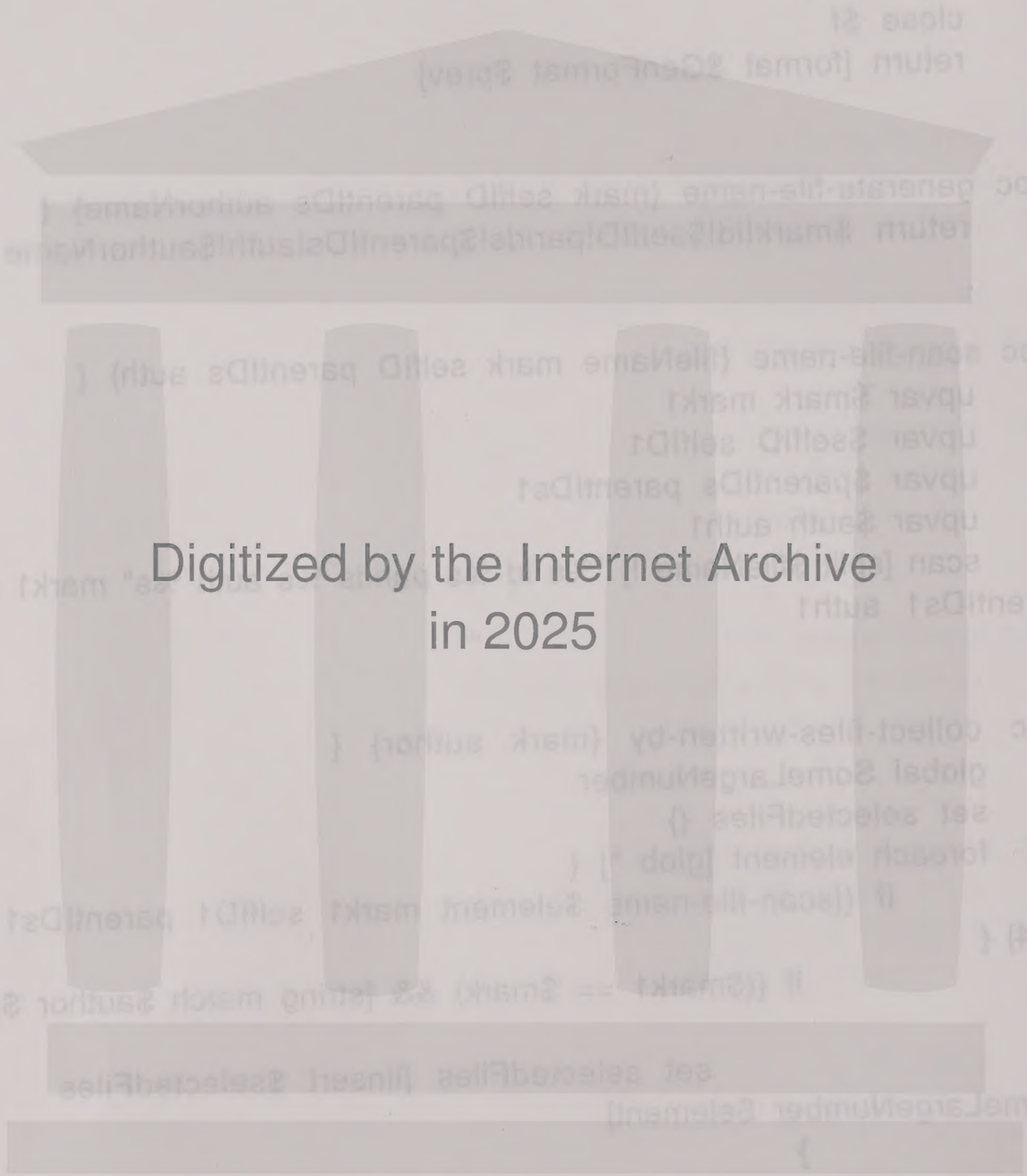
```
        }
        set f [open $seqNoFileName w]
        set prev [expr $prev + 1]
        puts $f $prev
        close $f
        return [format $GenFormat $prev]
}

proc generate-file-name {mark selfID parentIDs authorName} {
        return $mark!id!$selfID!parids!$parentIDs!auth!$authorName
}

proc scan-file-name {fileName mark selfID parentIDs auth} {
        upvar $mark mark1
        upvar $selfID selfID1
        upvar $parentIDs parentIDs1
        upvar $auth auth1
        scan [split $fileName !] "%s id %s parids %s auth %s" mark1 selfID1
parentIDs1 auth1
}

proc collect-files-written-by {mark author} {
        global SomeLargeNumber
        set selectedFiles {}
        foreach element [glob *] {
                if {[scan-file-name $element mark1 selfID1 parentIDs1 auth1]
== 4} {
                        if {($mark1 == $mark) && [string match $author $auth1 ]}
{
                                set selectedFiles [linsert $selectedFiles
$SomeLargeNumber $element]
                        }
                }
        }
        return $selectedFiles
}

proc collect-files-of-mark {mark} {
        return [collect-files-written-by $mark *]
}
```



Digitized by the Internet Archive
in 2025

https://archive.org/details/programbymrokodo00tedn

```
proc collect-files-whose-id-is-one-of {mark IDs} {
    global SomeLargeNumber
    set selectedFiles {}
    foreach element [glob *] {
        if {[scan-file-name $element mark1 selfID1 parentIDs1 auth1]
== 4} {
            if {($mark == $mark1) &&(($IDs == $selfID1) || ([lsearch -
exact $IDs $selfID1] >= 0))} {
                set selectedFiles [linsert $selectedFiles
$SomeLargeNumber $element]
            }
        }
    }
    return $selectedFiles
}
proc collect-files-whose-parent-id-contains {mark parentID} {
    global SomeLargeNumber
    set selectedFiles {}
    foreach element [glob *] {
        if {[scan-file-name $element mark1 selfID1 parentIDs1 auth1]
== 4} {
            if {($mark == $mark1) && (($parentID == $parentIDs1) ||
([lsearch -exact [split $parentIDs1 :] $parentID] >= 0))} {
                set selectedFiles [linsert $selectedFiles
$SomeLargeNumber $element]
            }
        }
    }
    return $selectedFiles
}

proc sort-files-by-generation {fileNames} {
    global SomeLargeNumber
    set list {}
    foreach element $fileNames {
        scan-file-name $element mark1 selfID1 parentIDs1 auth1
        set list [linsert $list $SomeLargeNumber $selfID1]
    }
```

```
	set sortedList [lsort $list]
	set sortedFileNames {}
	foreach element $sortedList {
		set sortedFileNames [linsert $sortedFileNames
$SomeLargeNumber [lindex $fileNames [lsearch $list $element]]]
	}
	return $sortedFileNames
}

proc FILE:get-descriptor {fileName} {
	set f [open $fileName r]
	set descriptor [gets $f]
	close $f
	return $descriptor
}

proc FILE:get-name {fileName} {
	return [get-attribute [FILE:get-descriptor $fileName] name +]
}

proc FILE:get-chaned-name-descriptor {fileName newName} {
	set descriptor [FILE:get-descriptor $fileName]
	return [set-attribute $descriptor name $newName +]
}
```

zipplex.tcl
A program by Mr. Okidoki to:

```
proc select-zip-files-contains {zipFileNames itemFileName} {
      global SomeLargeNumber
      set selectedZips {}
      foreach zipFileName $zipFileNames {
            if {[lsearch [ZIP:get-item-list $zipFileName] $itemFileName] >=
0} {
                  set selectedZips [linsert $selectedZips
$SomeLargeNumber $zipFileName]
            }
      }
      return $selectedZips
}

proc select-all-zip-files-contains {itemFileName} {
      select-zip-files-contains [collect-files-of-mark ZIP]
$itemFileName
}

proc print-zip-files-content {zipFileNames} {
      foreach zipFileName $zipFileNames {
            puts -----$zipFileName-----
            ZIP:print-content $zipFileName
      }
}

proc ZIPPLEX:create {name itemList} {
      ZIP:create $name $itemList ZIPPLEX
}

proc ZIPPLEX:version-up {parentZipFileNames} {
      ZIP:version-up $parentZipFileNames ZIP
}

proc ls-zipplex {} {
```

```
    ls-zip ZIPPLEX
}

proc find-zipplex-file-named {name} {
    find-zip-file-named $name ZIPPLEX
}
```

```
zip.tcl
A program by Mr. Okidoki to:



set  EndOfItems  "*****"

proc  ZIP:get-item-list  {zipFileName}  {
      global  SomeLargeNumber
      global  EndOfItems
      set f [open $zipFileName r]
      set  descriptor  [gets  $f]
      set  itemList  {}
      while {! [eof $f]} {
            set  line  [gets  $f]
            if {$line == $EndOfItems} {
                  break;
            }
            set  itemList  [linsert  $itemList  $SomeLargeNumber  $line]
      }
      close  $f
      return  $itemList
}

proc ZIP:print {zipFileName} {
      foreach  item  [ZIP:get-item-list  $zipFileName]  {
            puts   -----$item-----
            ITEM:print  $item
      }
}
proc  ZIP:print-content  {zipFileName}  {
      foreach  item  [ZIP:get-item-list  $zipFileName]  {
            puts  $item
      }
}

proc  make-zip-from-file  {fileName}  {
      global  SomeLargeNumber
```

```
        set itemList {}
        set f [open $fileName r]
        set zip [ZIP:create $fileName {}]
        set newFileFlag 0
        set content ""
        while {! [eof $f]} {
                set line [gets $f]
                if {($line == "")} {
                        if {$newFileFlag} {
                                set item [ITEM:create $content _ $zip]
                                set itemList [linsert $itemList $SomeLargeNumber
$item]
                                set newFileFlag 0
                                set content ""
                        }
                } else {
                        if {! $newFileFlag} {
                                set newFileFlag 1
                        }
                        append content $line
                        append content "
"
                }
        }
        set zip [ZIP:level-up $zip $itemList]
        close $f
        return $zip
}

proc ZIP:set-name {fileName newName} {
        ZIP:create-internal $fileName [FILE:get-chaned-name-descriptor
$fileName $newName] [ZIP:get-item-list $fileName]
}

proc ZIP:create-internal {zipFileName descriptor itemList} {
        global EndOfItems
        set f [open $zipFileName w+]
        puts $f $descriptor
        foreach item $itemList {
                puts $f $item
```

```
        }
        puts $f $EndOfItems
        close $f
        return $zipFileName
}

proc ZIP:create {name itemList {mark ZIP}} {
        set zipFileName [generate-file-name $mark [generate-seq-no]
[generation-zero] [author-name]]
        return [ZIP:create-internal $zipFileName DESCRIPTOR+name+$name
$itemList]
}

proc ZIP:level-up {zipFileName itemList} {
        return [ZIP:create-internal $zipFileName [FILE:get-descriptor
$zipFileName] $itemList]
}

proc ZIP:version-up {parentZipFileNames {mark ZIP}} {
        global SomeLargeNumber
        if {[llength parentZipFileNames] == 0} {
                error "ZIP:version-up needs parentZipFileNames"
        }
        set parentIDs {}
        foreach element $parentZipFileNames {
                scan-file-name $element mark1 selfID1 parentIDs1 auth1
                set parentIDs [linsert $parentIDs $SomeLargeNumber $selfID1]
        }
        set descriptor [FILE:get-descriptor [lindex $parentZipFileNames 0]]
        set zipFileName [generate-file-name $mark [generate-seq-no]  [join
$parentIDs ":"] [author-name]]
        set itemList {}
        foreach element $parentZipFileNames {
                foreach item [ZIP:get-item-list $element] {
                        set itemList [linsert $itemList $SomeLargeNumber
$item]
                }
        }
        return [ZIP:create-internal $zipFileName $descriptor $itemList]
}
```

```
proc ls-zip {{mark ZIP}} {
	foreach element [collect-files-of-mark $mark] {
		puts [FILE:get-name $element]
	}
}

proc find-zip-file-named {name {mark ZIP}} {
	global SomeLargeNumber
	set selectedFiles {}
	foreach element [collect-files-of-mark $mark] {
		if [string match $name [FILE:get-name $element]] {
			set selectedFiles [linsert $selectedFiles
$SomeLargeNumber $element]
		}
	}
	return $selectedFiles
}

proc ITEMLIST:rearrange {itemList srcPos srcItem destPos} {
	if {$destPos < 0} {
		return [lreplace $itemList $srcPos $srcPos]
	}
	if {$srcPos < 0} {
		set newList [linsert $itemList $destPos $srcItem]
	} else {
		set newList [lreplace $itemList $srcPos $srcPos [lindex
$itemList $destPos]]
		set newList [lreplace $newList $destPos $destPos $srcItem]
	}
	return $newList
}

proc ZIP:rearrange-internal {zipFileName itemList srcPos srcItem
destPos} {
	ZIP:level-up $zipFileName [ITEMLIST:rearrange $itemList $srcPos
$srcItem $destPos]
}

proc ZIP:rearrange-item {zipFileName itemList srcPos destPos} {
```

```
    ZIP:rearrange-internal $zipFileName $itemList $srcPos [lindex
$itemList $srcPos] $destPos
}

proc ZIP:delete-item {zipFileName itemList pos} {
    set item [lindex $itemList $pos]
    ZIP:rearrange-internal $zipFileName $itemList $pos $item -1
    return $item
}

proc ZIP:insert-item {zipFileName itemList item pos} {
    ZIP:rearrange-internal $zipFileName $itemList -1 $item $pos
    return $item
}

proc ITEMLIST:replace {itemList pos item} {
    if {$pos < 0} {
        error "(ITEMLIST:replace) no such item: $item"
    }
    return [lreplace $itemList $pos $pos $item]
}

proc ZIP:replace-item {zipFileName itemList pos item} {
    ZIP:level-up $zipFileName [ITEMLIST:replace $itemList $pos $item]
    return $item
}

proc ZIP:update-item {zipFileName itemList pos content} {
    return [ZIP:replace-item $zipFileName $itemList $pos [ITEM:update
[lindex $itemList $pos] $content]]
}
```

xx.tcl
A program by Mr. Okidoki to:

```
proc find-file {pat} {
      set new {}
      foreach el [exec ls] {
      if [string match $pat $el] {
            lappend new $el
      }
      }
      return $new
}

proc get-ctime {file} {
      file stat $file info
      return $info(ctime)
}

# very silly
proc get-current-time {} {
      set dummyFileName .dummy
      set f [open $dummyFileName r]
      close $f
      file stat $dummyFileName info
      return $info(ctime)
}

proc sort-file {fileName} {
      set sorted {}
      set f [open $fileName r]
      while {! [eof $f]} {
            lappend sorted [gets $f]
      }
      close $f
      lsort $sorted
}
```

```
proc print-file2 {fileName} {
    set f [open $fileName r]
    while {! [eof $f]} {
        set line [gets $f]
        if {$line == ""} {
            puts ----------
        } else {
            puts $line
        }
    }
    close $f
}

proc generate-timestamp {creationTime modificationTime} {
    if {$creationTime == 0} {
        set creationTime [get-current-time]
        set modificationTime $creationTime
    } elseif {$modificationTime == 0} {
        set modificationTime [get-current-time]
    }
    return !ctime!$creationTime!mtime!$modificationTime
}

proc sort-files-with-modification-time {fileNames} {
    global SomeLargeNumber
    set list {}
    foreach element $fileNames {
        scan-file-name $element mark1 auth1 anno1 ctime1 mtime1
        set list [linsert $list $SomeLargeNumber $mtime1]
    }
    set sortedList [lsort -integer $list]
    set sortedFileNames {}
    foreach element $sortedList {
        set sortedFileNames [linsert $sortedFileNames
$SomeLargeNumber [lindex $fileNames [lsearch $list $element]]]
    }
    return $sortedFileNames
}
```

```
proc collect-files1 {fileName} {
    global SomeLargeNumber
    scan-file-name $fileName fileName mark selfID parentIDs auth
    set selectedFiles {}
    foreach element [glob *] {
        if {[scan-file-name $element mark1 selfID1 parentIDs1 auth1]
== 4} {
            if {[expr {($mark == "*") ?  1 : ($mark == $mark1)}] == 0} {
                continue
            }
            if {[expr {($selfID == "*") ?  1 : ($selfID == $selfID1)}] ==
0} {
                continue
            }
            if {[expr {($parentIDs == "*") ?  1 : ($parentIDs ==
$parentIDs1)}] == 0} {
                continue
            }
            if {[expr {($auth == "*") ?  1 : ($auth == $auth1)}] == 0} {
                continue
            }
            set selectedFiles [linsert $selectedFiles
$SomeLargeNumber $element]
        }
    }
    return $selectedFiles
}
```

```
test.tcl
A program by Mr. Okidoki to:

rm .seqNo
rm ZIP*
rm ITEM*

tclsh

--------------------

source prog/init.tcl

set heretic [make-zip-from-file heretic.txt]

set plex [ZIPPLEX:create plex $heretic]

ls-zip
ls-zipplex

cat [find-zip-file-named heretic.txt]

ZIP:print $heretic

set heretic [find-zip-file-named heretic.txt]

set item [ZIP:update-item $heretic [ZIP:get-item-list $heretic] 1 "abcde"]

cat $heretic

set item [ITEM:find-parents $item]
set item [ITEM:find-children $item]

set item [ZIP:insert-item $heretic [ZIP:get-item-list $heretic]
[ITEM:create "abcdefg" "test" $heretic] 3]

cat $heretic
```

```
ITEM:get-home-zip  $item

set  itemList  [ZIP:get-item-list  $heretic]
set item [ZIP:insert-item $heretic $itemList [ITEM:update [list [lindex
$itemList 2] [lindex $itemList 3]] "1234567"] 3]

ITEM:find-parents  $item

cat  $heretic

ZIP:delete-item  $heretic  [ZIP:get-item-list  $heretic]    4

cat  $heretic

ZIP:rearrange-item  $heretic  [ZIP:get-item-list  $heretic]    1  2

cat  $heretic

set  zip  [ZIP:version-up  $heretic]

cat  $zip

find-zip-file-named   heretic.txt

sort-files-by-generation   [find-zip-file-named   heretic.txt]

ZIP:set-name $zip GGG

ls-zip

select-all-zip-files-contains   $item
```

random.tcl
A program by Mr. Okidoki to:

```
proc random {} {
	global rnseed
	set rnseed [expr 30903*($rnseed&65535)+($rnseed>>16)]
	return [expr ($rnseed & 65535)/65535.0]
}

#Use this to initialize the random number:
proc initrand {} {
	global rnseed
	set rnseed [exec sh -c "echo $$"]
}
```

item.tcl
A program by Mr. Okidoki to:

```
proc ITEM:create-internal {parentIDs content name homeZipFileName } {
	set itemFileName [generate-file-name ITEM [generate-seq-no]
$parentIDs [author-name]]
	set f [open $itemFileName w]
	puts $f DESCRIPTOR+name+$name+zip+$homeZipFileName
	puts $f $content
	close $f
	return $itemFileName
}
proc ITEM:update-internal {parentItemFileNames content name
homeZipFileName} {
	global SomeLargeNumber
	set parentIDs {}
	foreach element $parentItemFileNames {
		scan-file-name $element mark1 selfID1 parentIDs1 auth1
		set parentIDs [linsert $parentIDs $SomeLargeNumber $selfID1]
	}
	if {$parentIDs == ""} {
		set parentIDs [generation-zero]
	} else {
		set parentIDs [join $parentIDs ":"]
		if {([string match $name _]) || ([string match $homeZipFileName
_])} {
			set representativeParentItem [lindex
$parentItemFileNames 0]
			set descriptor [FILE:get-descriptor
$representativeParentItem]
			if {[string match $name _]} {
				set name [get-attribute $descriptor name +]
			}
			if {[string match $homeZipFileName _]} {
				set homeZipFileName [get-attribute $descriptor zip
+]
			}
```

```
            }
        }
        return [ITEM:create-internal $parentIDs $content $name
$homeZipFileName]
}

proc ITEM:get-home-zip {fileName} {
        return [get-attribute [FILE:get-descriptor $fileName] zip +]
}

proc ITEM:update {parentItemFileNames content} {
        ITEM:update-internal $parentItemFileNames $content _ _
}

proc ITEM:create {content {name _} {homeZipFileName _}} {
        return [ITEM:update-internal "" $content $name $homeZipFileName]
}

proc ITEM:find-parents {itemFileName} {
        scan-file-name $itemFileName mark1 selfID1 parentIDs1 auth1
        return [collect-files-whose-id-is-one-of ITEM [split $parentIDs1 :]]
}

proc ITEM:find-children {itemFileName} {
        scan-file-name $itemFileName mark1 selfID1 parentIDs1 auth1
        return [ collect-files-whose-parent-id-contains ITEM $selfID1]
}

proc ITEM:content {itemFileName} {
        set f [open $itemFileName r]
        set descriptor [gets $f]
        set content ""
        while {! [eof $f]} {
                set line [gets $f]
                if {($line == "")} {
                } else {
                        append content $line
                        append content "
"
                }
```

```
        }
        close $f
        return $content
}

proc ITEM:print {itemFileName} {
        set f [open $itemFileName r]
        set descriptor [gets $f]
        set cond 1
        while {$cond} {
                set line [gets $f]
                if {$line != ""} {
                        puts -nonewline $line
                }
                if {! [eof $f]} {
                        puts ""
                } else {
                        set cond 0
                }
        }
        close $f
}
```

init.tcl
A program by Mr. Okidoki to:

```
source  prog/com.tcl
source  prog/array.tcl
source  prog/file.tcl
source  prog/item.tcl
source  prog/zip.tcl
source  prog/zipplex.tcl
```

gen-news-html.tcl
A program by Mr. Okidoki to:

```
proc generate-news-group-html {inFileName outFileName} {
        set f [open $inFileName r]
        set g [open $outFileName w]
        puts $g <ul>
        while {! [eof $f]} {
                set newsGroupName [lindex [split [gets $f] :] 0]
                puts $g <li><a\
href="news:$newsGroupName">$newsGroupName</a>
        }
        puts $g </ul>
        close $g
        close  $f
}
```